

[Inicio](#) | [Fundamentos](#) | [Instalación](#) | [Añadir Elementos](#) | [Encadenar Métodos](#) | [Asociar Datos](#) | [Usando sus Datos](#) | [Desplegar DIVs](#) | [La Función data\(\)](#) | [Introducción a SVG](#) | [Despliegue de SVG](#) | [Tipos de Datos](#) | [Diagrama de Barras](#) | [Diagrama de Dispersión](#) | [Escala](#) | [Ejes](#) |

Cómo Hacer un Diagrama de Barras

Ahora vamos a integrar todo lo que hemos aprendido hasta el momento para generar un diagrama de barras simple con D3.

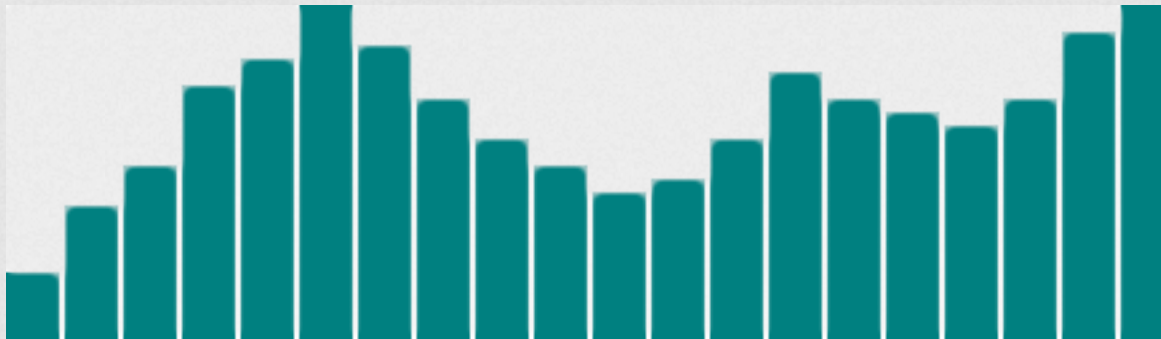
Empezamos revisando el diagrama de barras que hicimos anteriormente usando elementos `div`. Luego, vamos a adaptar ese código para dibujar las barras esta vez con SVG, lo cual nos da más flexibilidad en cuanto a la presentación visual. Por último, vamos a añadir etiquetas, de tal manera que podamos ver el valor de los datos claramente.

El Diagrama Anterior

Acá está lo que hicimos la última vez, con datos nuevos.

```
var dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13,  
11, 12, 15, 20, 18, 17, 16, 18, 23, 25 ];
```

```
d3.select("body").selectAll("div")
  .data(dataset)
  .enter()
  .append("div")
  .attr("class", "bar")
  .style("height", function(d) {
    var barHeight = d * 5;
    return barHeight + "px";
  });
```



Puede ser difícil imaginárselo, pero definitivamente podemos mejorar este diagrama hecho con `divs`.

El Nuevo Diagrama

Lo primero es lo primero - toca decidir el tamaño del nuevo SVG:

```
//Ancho y Altura  
var w = 500;  
var h = 100;
```

(Por supuesto, se debe nombrar `w` y `h` diferente, como `svgWidth` y `svgHeight`. Utilice lo que tenga más sentido para usted. JavaScript tiene una fijación cultural con la eficiencia, entonces es común encontrar variables de un solo caracter, código escrito sin espacios y otro tipo de sintaxis que es difícil de leer pero que es eficiente desde el punto de vista de programación.)

Luego, le decimos a D3 que cree un elemento SVG vacío y que lo añada al DOM:

```
//Crear un elemento SVG  
var svg = d3.select("body")  
    .append("svg")  
    .attr("width", w)  
    .attr("height", h);
```

Esto inserta un nuevo elemento `<svg>` inmediatamente antes de la etiqueta de cierre `</body>`, y le asigna el ancho de `w` y la altura de `h` de 500 por 100 pixeles. Esta cláusula también asigna el resultado a la nueva variable llamada `svg`, de tal manera que se pueda

referenciar fácilmente sin tener que volver a seleccionarla en el futuro usando algo así como `d3.select("svg")`.

Luego, en vez de crear `divs`, vamos a generar `rect` y los vamos a añadir a `svg`.

```
svg.selectAll("rect")
  .data(dataset)
  .enter()
  .append("rect")
  .attr("x", 0)
  .attr("y", 0)
  .attr("width", 20)
  .attr("height", 100);
```

Este código selecciona todos los `rect` dentro de `svg`. Por supuesto, aún no existen, entonces el programa devuelve una selección vacía. (Sí es raro, pero quédese y verá. Con D3 siempre es necesario seleccionar primero aquello sobre lo que va a aplicar una acción, aún cuando esa selección esté vacía por el momento.)

Luego, `data(dataset)` ve que existen 20 valores en el conjunto de datos, y llama 20 veces a `enter()`. `enter()` a su vez devuelve la posición a una selección por cada dato que hasta el momento no tiene un `rect` correspondiente – o sea, todos.

Para cada una de esas posiciones, la función `append("rect")` inserta la etiqueta `rect` en el DOM. Como aprendimos en la introducción a SVG, cada `rect` debe tener valores de `x,y,width` y `height`. Se utiliza `attr()` para añadir estos atributos a cada uno de los rectángulos que se acaban de crear.

Qué belleza, no?

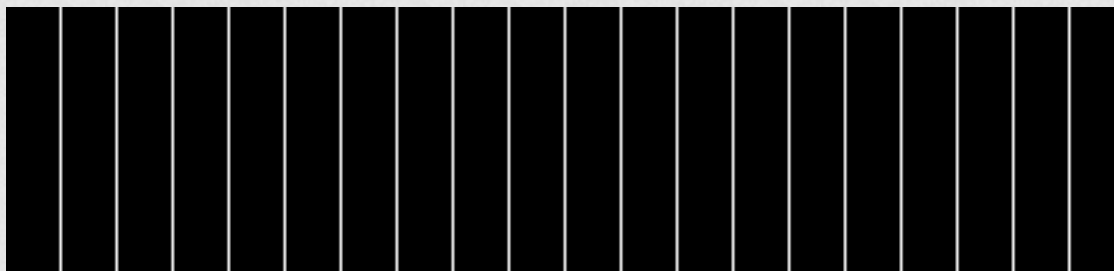


Bueno, tal vez no. Todas las barras están ahí (puede chequear el DOM de la [página de ejemplo](#), con el inspector de web), pero todas comparten los mismos valores de `x`, `y`, `width` y `height`, y el resultado es que todas quedan sobrepuestas. Esto aún no es una visualización de datos.

Arreglemos primero el tema de la sobre posición. En vez de `x` con cero, se le asignará un valor dinámico que corresponde a `i`, o la posición del valor dentro del conjunto de datos. Entonces, la primera barra tendrá cero, pero las siguientes se ubicarán en el `21`, luego el `42` y así en adelante.

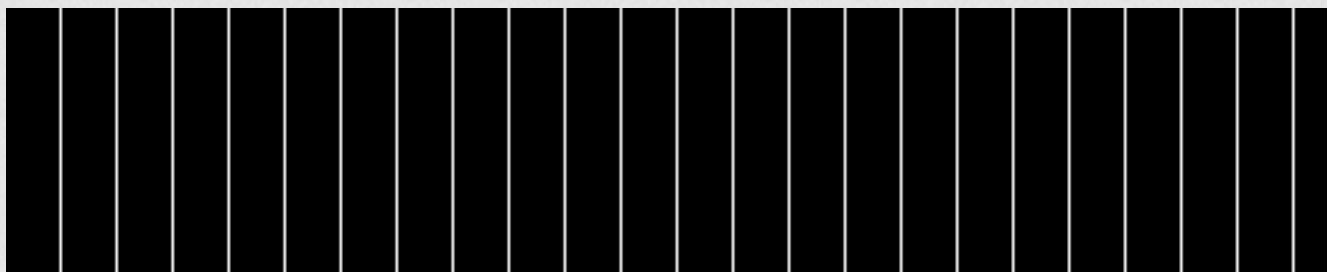
```
.attr("x", function(d, i) {
```

```
return i * 21; //Ancho de barras de 20 más 1 espacio  
})
```



Acá se ve el **código en acción**.

Esto funciona pero no es particularmente flexible. Si el conjunto de datos tuviera más elementos, las barras seguirían hacia la derecha y se saldrían del SVG! Debido a que cada barra tiene un ancho de 20 pixeles y uno adicional para presentación, el SVG de 500 pixeles solo puede incluir 23 datos. Mire cómo se corta la vigésima cuarta barra.



Es muy buena práctica utilizar coordenadas dinámicas y flexibles – alturas, anchos, valores x y y – para que la visualización pueda cambiar de escala de acuerdo con los

datos.

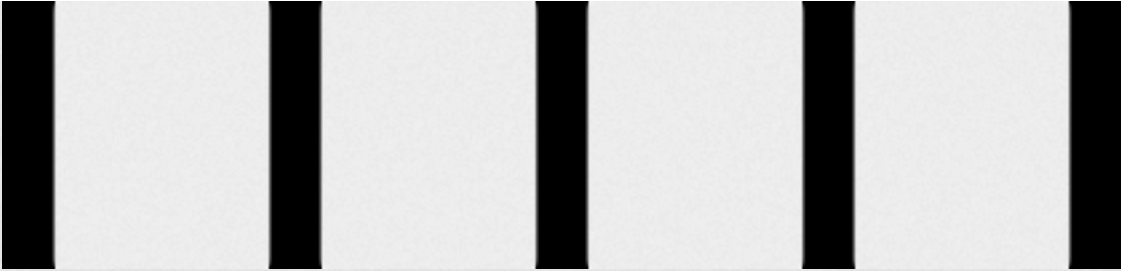
Como todo en programación, existen mil maneras de lograr ese objetivo. Acá se va a mostrar una sencilla. Primero, se cambia la línea donde se definió la posición x de cada barra.

```
.attr("x", function(d, i) {  
  return i * (w / dataset.length);  
})
```

Se puede apreciar cómo el valor de x ahora está directamente asociado al ancho del SVG (w) y al número total de valores del conjunto de datos (`dataset.length`). Esto es novedoso, porque ahora las barras están bien distribuidas, si se tienen 20 valores



o solo cinco:



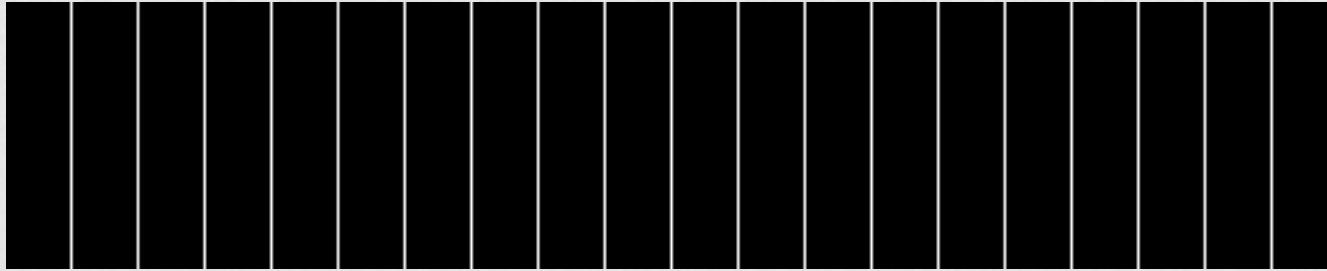
Acá está el código hasta el momento

Ahora se debe definir el ancho de las barras para que sean proporcionales, de tal manera que se vuelvan más angostas cuando se añaden más datos o se agranden cuando hayan menos valores. Voy a añadir una nueva variable cerca de donde se definen el ancho y la altura del SVG.

```
//Width and height  
var w = 500;  
var h = 100;  
var barPadding = 1; // <-- Nueva!
```

y luego se referencia esta variable en la línea donde se define el ancho de cada barra. En vez de tener un valor estático de 20, el ancho se define como una fracción del ancho del SVG y del número de datos, menos el valor del espacio de colchón (padding).

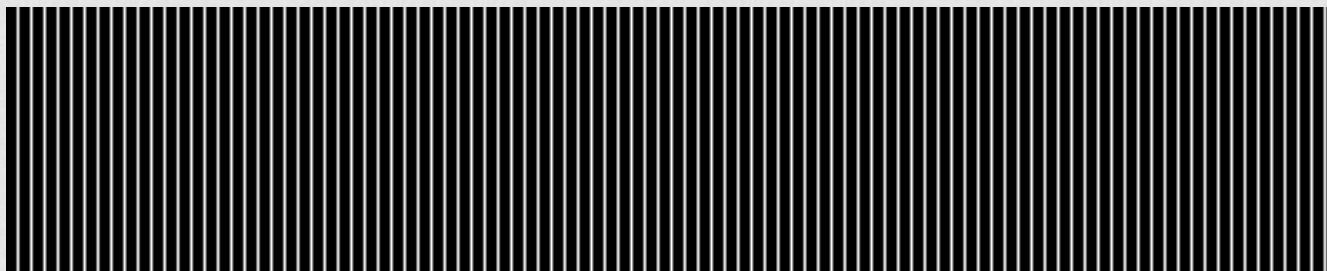

```
` .attr("width", w / dataset.length - barPadding)`
```



Funciona!. El ancho de las barras y la posición horizontal cambian de escala correctamente cuando hay 20 datos o solo cinco.



O también 100:



Por último, codificamos los datos de tal forma que correspondan a la *altura* de cada barra. Pensaría uno que es tan fácil como referenciar el valor del dato `d` para asignarle la altura (`height`) a cada barra:

```
.attr("height", function(d) {  
  return d;  
});
```



Se va raro. Tratamos de cambiar la escala de los número un poco?

```
.attr("height", function(d) {  
  return d * 4; // <-- Cuatro veces!  
});
```



Realmente no es así de fácil – se quiere que las barras crezcan hacia arriba desde el borde inferior, no hacia abajo desde el tope. No se le debe echar la culpa a D3, más bien a SVG.

Si recuerda del capítulo de introducción a SVG, cuando se dibujan `rect`, la `x` y la `y` se especifican las coordenadas de la esquina superior izquierda. Esto quiere decir que el origen o punto de referencia de cada `rect` es esa esquina. Para nuestro propósito, sería mucho más fácil definir el origen en la esquina inferior izquierda, pero así no es como funciona SVG, y en realidad a SVG no le interesa qué pensemos al respecto.

Dado que nuestras barras deben “crecer hacia abajo desde la parte superior,” dónde queda entonces “el tope” de cada barra con respecto al tope del SVG? El tope de cada barra se puede expresar como la relación entre la altura del SVG y el valor del dato correspondiente, de la siguiente manera:

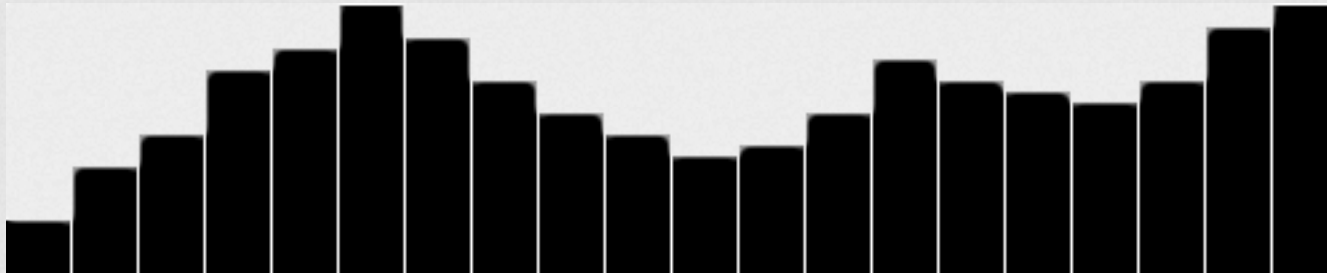
```
.attr("y", function(d) {  
    return h - d; //Altura menos el dato  
})
```

Ahora, para poner la parte inferior de la barra en el fondo del SVG, cada altura de `rect` puede ser el dato en sí.

```
.attr("height", function(d) {  
  return d; //Solo el dato  
});
```



Cambiamos la escala vertical multiplicando `d` por 4, o `d*4`. (Próximamente aprenderemos acerca de las *escalas* en D3, que tiene mejores maneras de lograr esto mismo.)

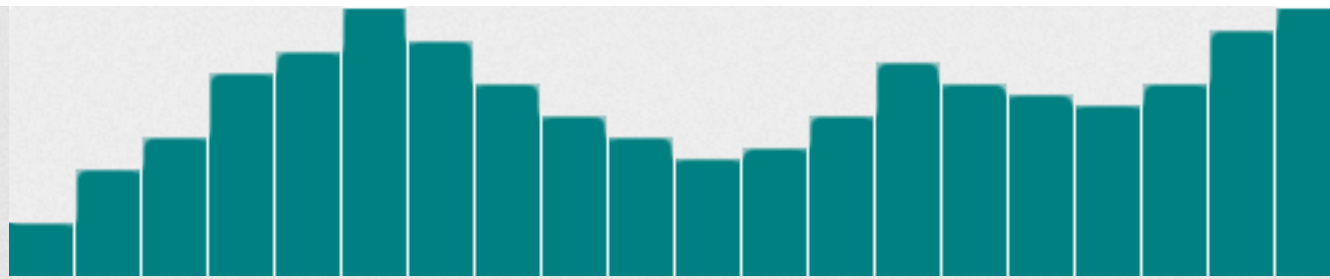


Acá está el **código de nuestro diagrama de barras** que está creciendo de arriba hacia abajo.

Color

Añadir color es fácil. Solamente utilice `attr()` y asigne un color de relleno `fill`:

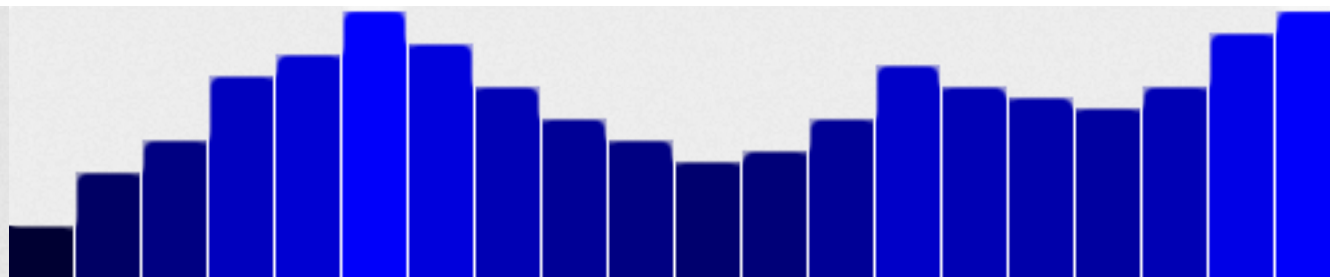
```
.attr("fill", "teal");
```

Acá está el **diagrama de barras verde azulado**. Usualmente se usa el color de la figura para mostrar algún otro atributo del dato. Esto quiere decir que se puede codificar un dato con el uso de colores. (En el caso de nuestro diagrama de barras, esto permite *codificación doble*, con lo cual un solo dato tiene dos tipos de códigos visuales: altura y color.)

Definir el color del gráfico a partir de los datos es muy fácil - se escribe una función que nuevamente referencia a `d`. Acá, se reemplaza el color verde azulado ("`teal`") con una función:

```
.attr("fill", function(d) {  
  return "rgb(0, 0, " + (d * 10) + ")";  
});
```



Acá está el código. No es particularmente útil desde el punto de vista visual, pero ya se empieza a tener la idea de cómo se puede traducir un dato a un color. Acá, `d` se multiplica por 10, y luego se usa el valor azul en la definición del color con la función `rgb()`. Con esto, a medida que el valor de `d` es mayor (más alta la barra), la barra toma un color más azul. A medida que es menor el valor, la barra tendrá menos azul (se verá más cerca del color negro).

Etiquetas

Las gráficas son excelentes pero con alguna frecuencia también es necesario desplegar los valores de los datos dentro de la gráfica. Acá es donde las etiquetas de datos juegan un papel y es muy, muy fácil desplegarlas con D3.

Si recuerda, en la introducción a SVG se vio cómo se puede añadir elementos de texto (`texto`) a un elemento de SVG. Se empieza con:

```
svg.selectAll("text")  
  .data(dataset)  
  .enter()
```

```
.append("text")
```

LO reconoce? De igual manera a como se hizo con `rect`, acá se hace con `text`. Primero, se selecciona lo que se necesita, se traen los datos, se crean nuevos elementos (que inicialmente son solamente espacios vacíos), y finalmente se añaden nuevos elementos `text` al DOM.

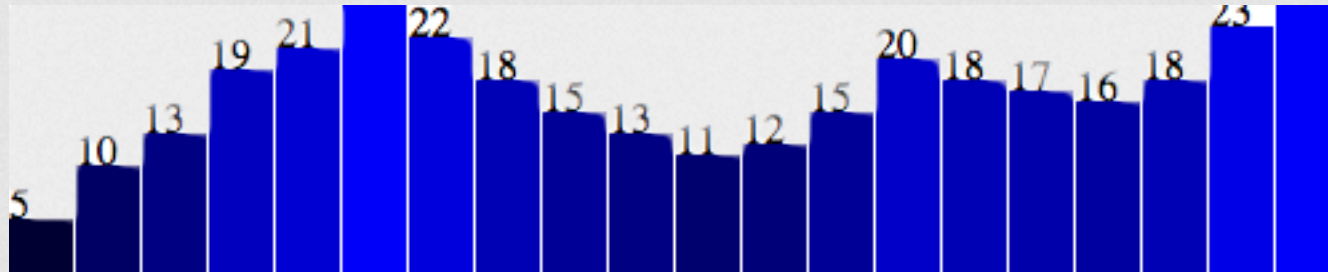
Vamos a añadir al código para incluir el valor del dato dentro de cada elemento `text`, por medio del método `text()`.

```
.text(function(d) {  
    return d;  
})
```

y luego se va a ampliar esto aún más, incluyendo los valores `x` y `y` para ubicar el texto. La forma más fácil consiste en copiar y pegar el mismo código de `x`/`y` que se usó antes para las barras.

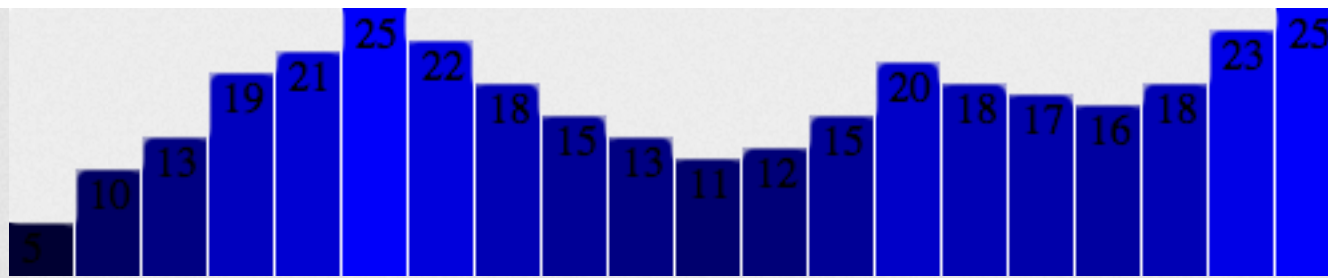
```
.attr("x", function(d, i) {  
    return i * (w / dataset.length);  
})
```

```
.attr("y", function(d) {
    return h - (d * 4);
});
```



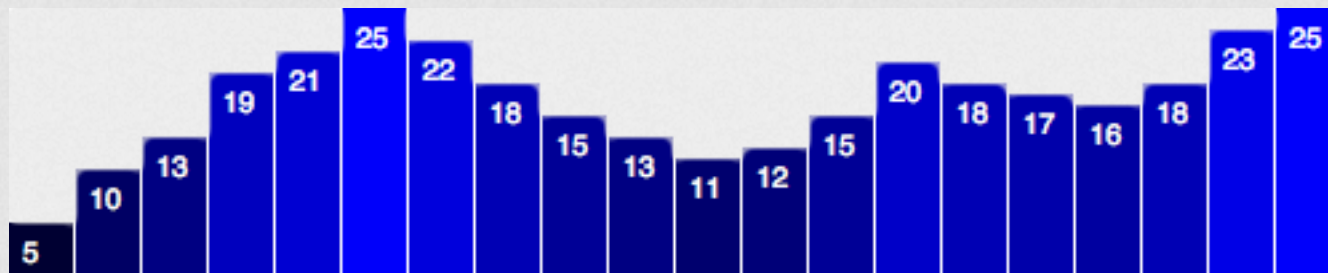
Ajá! Etiquetas para los datos! Pero algunos están cortados en la parte de arriba.
 Tratemos de moverlos hacia abajo hacia el interior de las barras, lo cual se puede hacer con solo añadir algunos cálculos para los valores de `x` y `y`.

```
.attr("x", function(d, i) {
    return i * (w / dataset.length) + 5; // +5
})
.attr("y", function(d) {
    return h - (d * 4) + 15;           // +15
});
```

Mejor, pero aún no es muy legible. Afortunadamente se puede corregir esto:

```
.attr("font-family", "sans-serif")  
.attr("font-size", "11px")  
.attr("fill", "white");
```

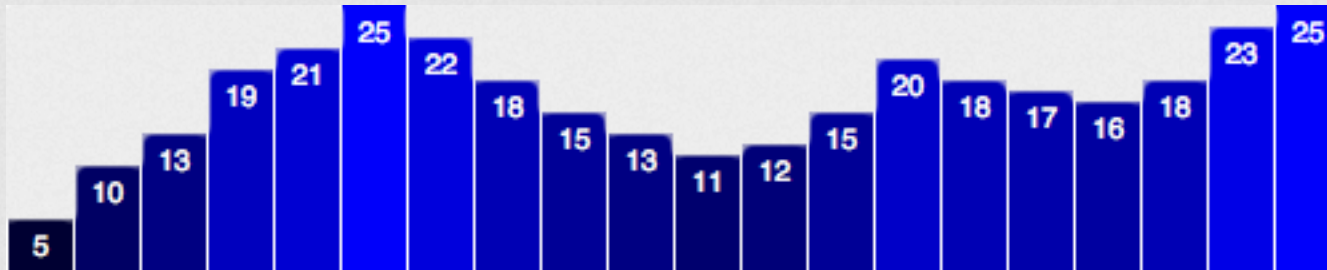


Fantásti-código! Si no vive obsesionado con la tipografía, ya terminó. Pero si es como yo, notará que las etiquetas no están alineadas perfectamente dentro de las barras. Eso se puede resolver muy fácilmente. Usemos el atributo `text-anchor` para centrar el texto horizontalmente sobre al valor de `x`.

```
` .attr("text-anchor", "middle")`
```

Ahora, cambiemos la manera como se calcula la posición `x` asignándole el borde izquierdo *más* la mitad del ancho de la barra:

```
.attr("x", function(d, i) {  
    return i * (w / dataset.length) + (w / dataset.length - barPadding) / 2;  
})
```



Completo!. Ya podemos salirnos de los diagramas de barras hacia otros temas.

Siguiente: Diagrama de Dispersión →

Este tutorial fue traducido por Gabriel Coch

Todo el contenido fue desarrollado por y le pertenece a Scott Murray

